IS 6733
Deep Learning on Cloud Platforms

Lecture 4b

Machine Learning Fundamentals – Part2

Dr. Yuanxiong Guo

Department of Information Systems and Cyber Security

Machine Learning Workflow

- Basic recipe for applying a supervised machine learning model
  1. Choose a class of model
  2. Choose model hyperparameters
  3. Fit the model to the training data
  4. Use the model to predict labels for new data
- The first two pieces are perhaps the most important part of using these tools and techniques effectively.
- To make an informed choice, we need a way to *validate* that our model and our hyperparameters are a good fit to the data.

Model Validation

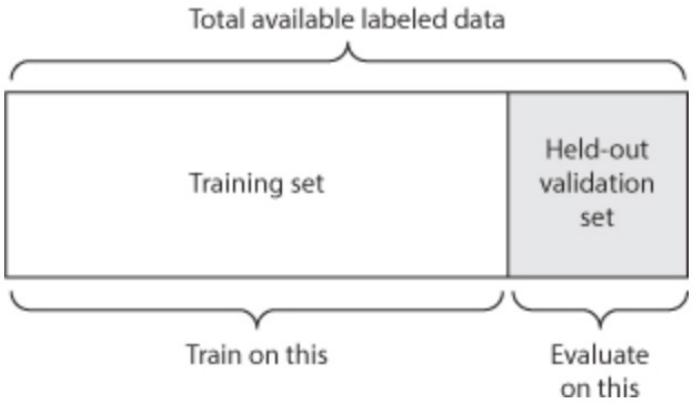
- How effective is it after choosing a model and its hyperparameters?

- Common mistake: train and evaluate the model on the same data
  - Code example: kNN model with k = 1

Model Validation: Holdout Sets

- Hold back some subset of the data from the training of the model, and then use this holdout set to check the model performance

Figure 4.1. Simple hold-out validation split



- Data spliting using the `train_test_split` utility in Scikit-Learn

```
from sklearn.model_selection import train_test_split
# split the data with 50% in each set
X1, X2, y1, y2 = train_test_split(X, y, random_state=0,
train_size=0.5)
```

Model Validation: Holdout Sets

- Weakness: we have lost a portion of the data to the model training
  - Can cause problem if the initial set of training data is small
  - The training, validation and testing datasets may contain too few samples to be statistically representative of the data at hand

- Solution: do a sequence of fits where each subset of the data is used both as a training set and as a validation set

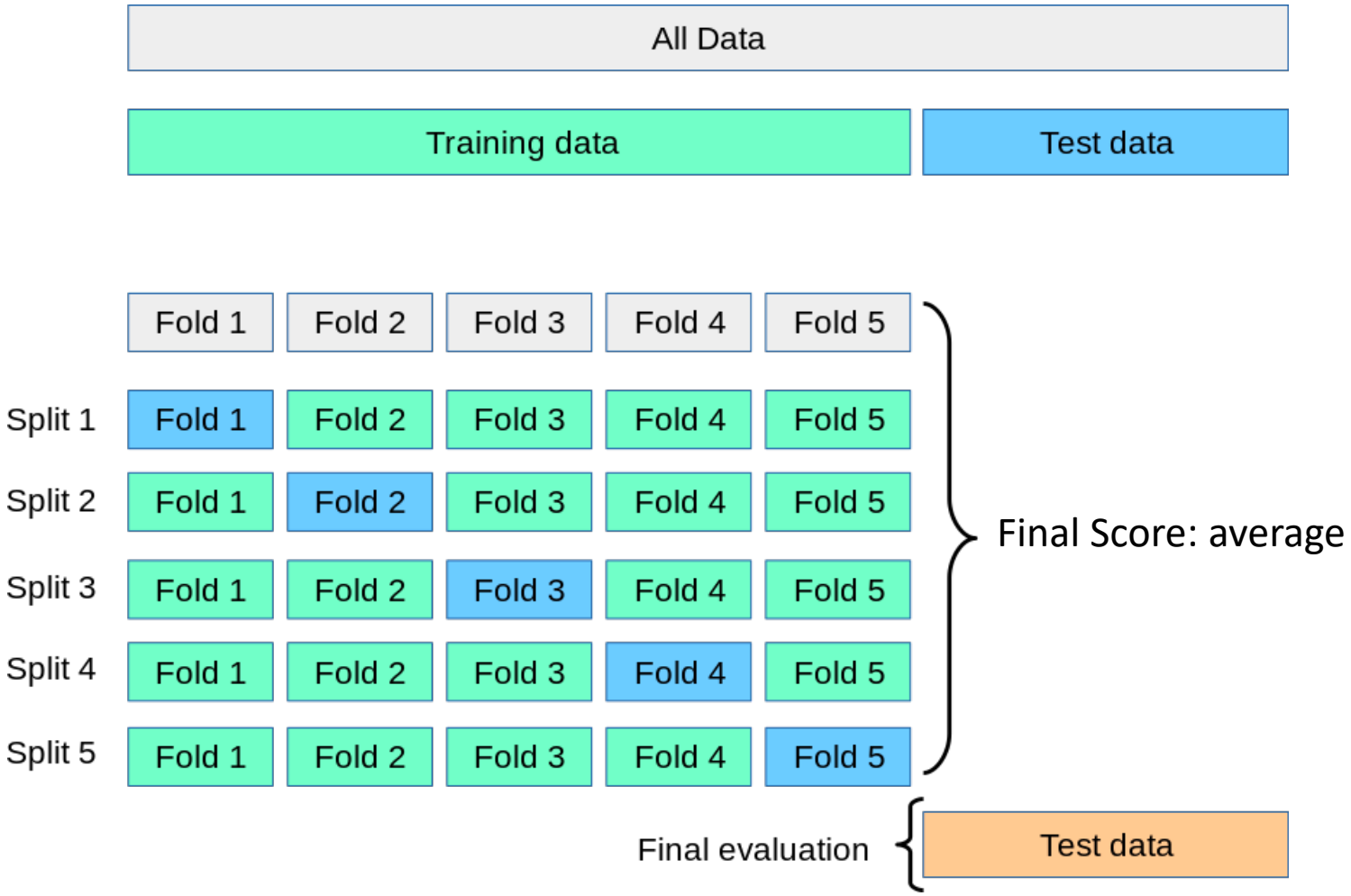
Model Validation: Cross-Validation

- Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data.
- Use cross-validation to detect overfitting (i.e, failing to generalize a pattern) and do model selection (i.e., choosing appropriate model hyperparameters).

K-fold Cross-Validation

- In K-fold cross-validation, you split the input training data into K subsets of data (also known as folds).
- You train an ML model on all but one (K-1) of the subsets, and then evaluate the model on the subset that was not used for training.
- This process is repeated K times, with a different subset reserved for evaluation (and excluded from training) each time.
- The final score is the average of the K scores obtained

5-fold Cross-Validation



Cross-Validation in Scikit-Learn

- The simplest way to use cross-validation is to call the `cross_val_score` function on the estimator and the dataset.

```
from sklearn.model_selection import cross_val_score

cross_val_score(model, X, y, cv=5)
```

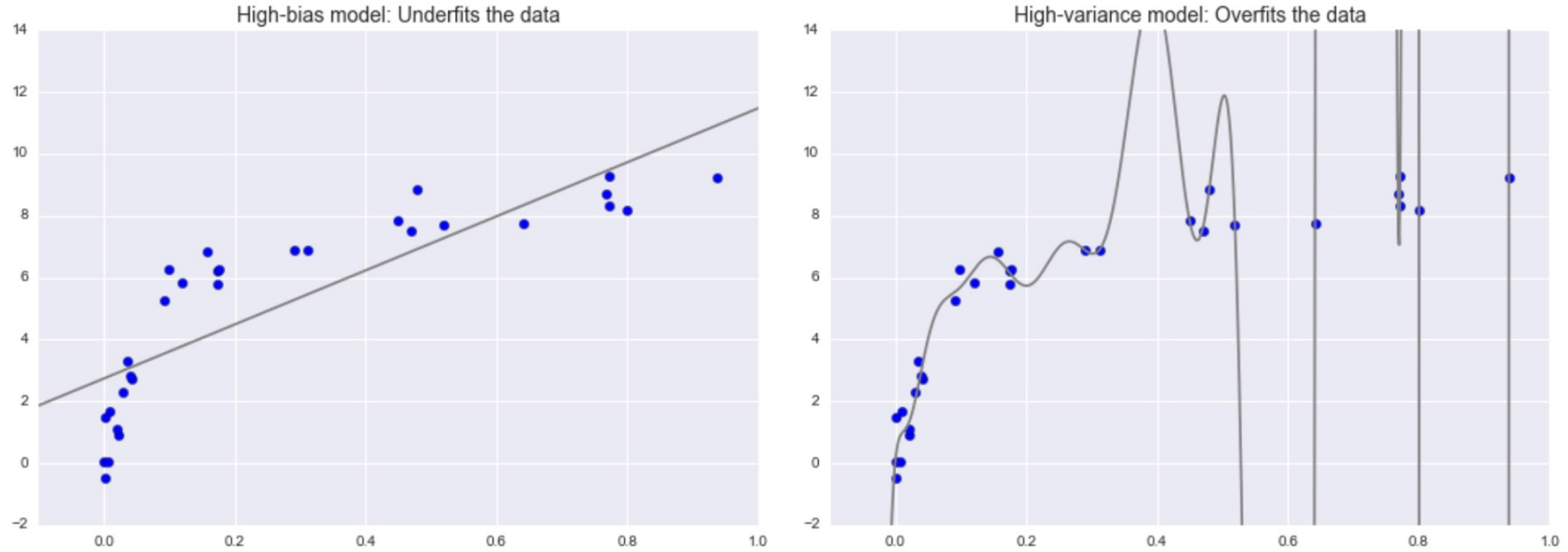
- Code example on validation

Model and Hyperparameter Selection

- If our estimator is underperforming, how should we move forward?
  - Use a more complicated/more flexible model
  - Use a less complicated/less flexible model
  - Gather more training samples
  - Gather more data to add features to each sample

Overfitting and Underfitting

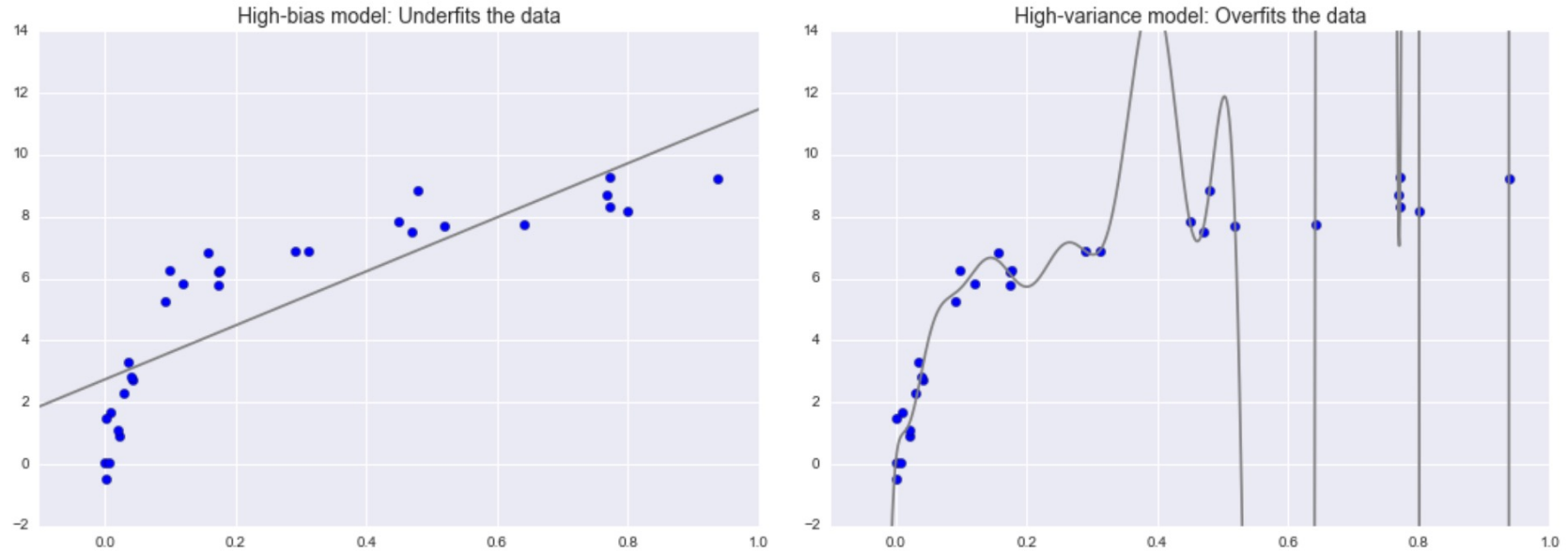
- Two regression fits to the same dataset



- The model on the left attempts to find a straight-line fit through the data, which are intrinsically more complicated than a straight line.
- The model is said to *underfit the data*: it does not have enough model flexibility to suitably account for all the features in the data.
- Another way of saying this is that the model has high bias.

Overfitting and Underfitting

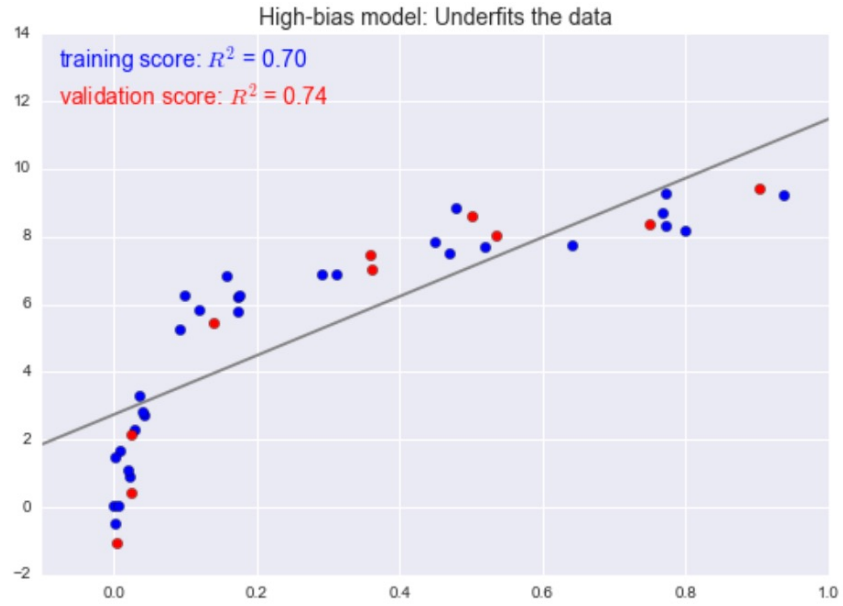
- Two regression fits to the same dataset

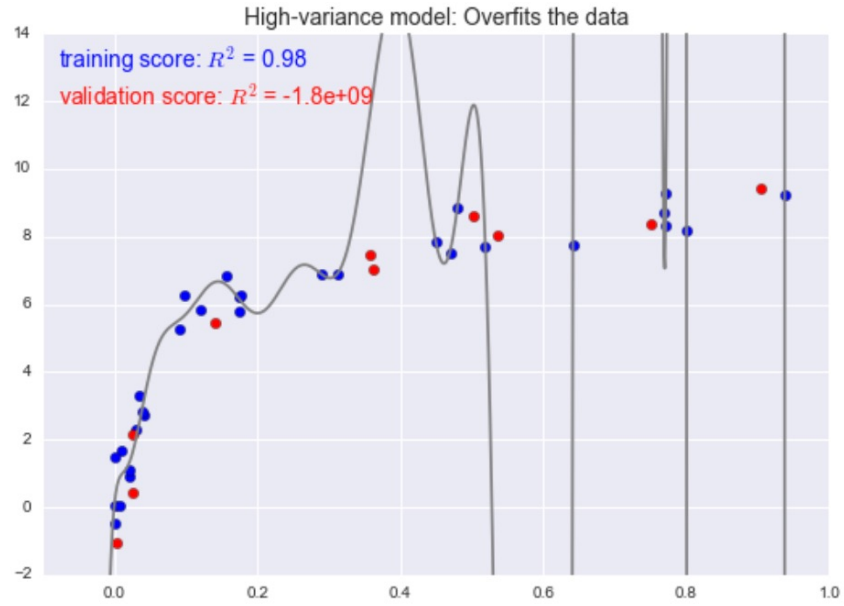


- The model on the right attempts to fit a high-order polynomial through the data, which very accurately describes the training data but its precise form seems to be more reflective of the particular noise properties of the data
- The model is said to *overfit the data*: it has so much model flexibility that the model ends up accounting for random errors as well as the underlying data distribution
- Another way of saying this is that the model has high variance.

Overfitting and Underfitting

- Predict the y-value for some new data

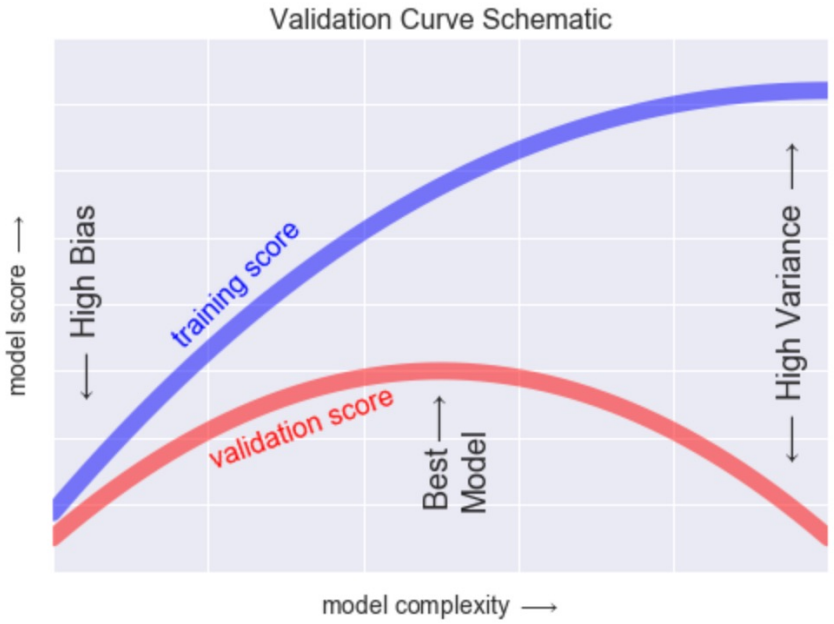




- For underfitting, the performance of the model on the validation set is similar to the performance on the training set.
- For overfitting, the performance of the model on the validation set is far worse than the performance on the training set.

Validation Curve

- If we imagine that we have some ability to tune the model complexity



- The means of tuning the model complexity varies from model to model
  - polynomial regression → the degree of the polynomial
  - neural networks → the number/size of layers

Validation in Scikit-Learn

- Given a model, data, parameter name, and a range to explore, `validation_curve` will automatically compute both the training score and validation score across the range

- In practice, Scikit-Learn provides automated tools to find the best hyperparameters in the grid search module: `GridSearchCV`

- Code example on validation curve and grid search

Data Preprocessing

- All of the examples before assume that you have numerical data in a [n_samples, n_features] format.
- In the real world, data rarely comes in such a form: image, text, audio, etc..
- Data preprocessing aims at making the raw data more amenable to machine learning models
  - Vectorization
  - Value normalization
  - Handling missing values
  - Feature extraction
- Many of these techniques are domain specific

Vectorization

- Convert arbitrary data into well-behaved vectors
- Categorical data: one-hot encoding

```
data = [
{'price': 850000, 'rooms': 4, 'neighborhood': 'Queen Anne'},
{'price': 700000, 'rooms': 3, 'neighborhood': 'Fremont'},
{'price': 650000, 'rooms': 3, 'neighborhood': 'Wallingford'},
{'price': 600000, 'rooms': 2, 'neighborhood': 'Fremont'}
]
from sklearn.feature_extraction import DictVectorizer
vec = DictVectorizer(sparse=False, dtype=int)
vec.fit_transform(data)
```

```
array([[ 0,  1,  0, 850000,  4],
       [ 1,  0,  0, 700000,  3],
       [ 0,  0,  1, 650000,  3],
       [ 1,  0,  0, 600000,  2]])
```

Another tool: `sklearn.preprocessing.OneHotEncoder`

Vectorization

- Text data: *word counts*
  - Take each snippet of text, count the occurrences of each word within it, and put the results in a table

```
sample = ['problem of evil',
'evil queen',
'horizon problem']
```

```
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer()
X = vec.fit_transform(sample)

import pandas as pd
pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

	evil	horizon	of	problem	queen
0	1	0	1	1	0
1	1	0	0	0	1
2	0	1	0	1	0

Value Normalization

- To make learning easier for your model, the data often should have the following characteristics:
  - Take small values – typically, most values would be in the 0-1 range
  - Be homogeneous – all features should take values in roughly the same range
- Additionally, common to have stricter normalization in practice
  - Normalize each feature independently to have a mean of 0
  - Normalize each feature independently to have a standard deviation of 1

```
x -= x.mean(axis = 0)
x /= x.std(axis =0)
```

Assume `x` is a 2D data matrix of shape (samples, features)

Handling Missing Values

- NaN value is used to mark missing values in NumPy
- When applying a typical machine learning model to such data, we will need to first replace such missing data with appropriate fill value (*imputation*)
- Different strategies depend on specific applications
- For a baseline imputation approach, using the *mean*, *median*, or *most frequent* value, Scikit-Learn provides the *Imputer* class

```
from numpy import nan
X = np.array([[ nan, 0, 3 ],
[ 3, 7, 9 ],
[ 3, 5, 2 ],
[ 4, nan, 6 ],
[ 8, 8, 1 ]])
from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy='mean')
X2 = imp.fit_transform(X)
```

```
array([[4.5, 0. , 3. ],
[3. , 7. , 9. ], [3. ,
5. , 2. ], [4. , 5. ,
6. ], [8. , 8. , 1. ]])
```

Feature Engineering

- Use your own knowledge about the data and about the machine-learning algorithm at hand to make the algorithm work better by applying hardcoded (nonlearned) transformations to the data before it goes into the model.

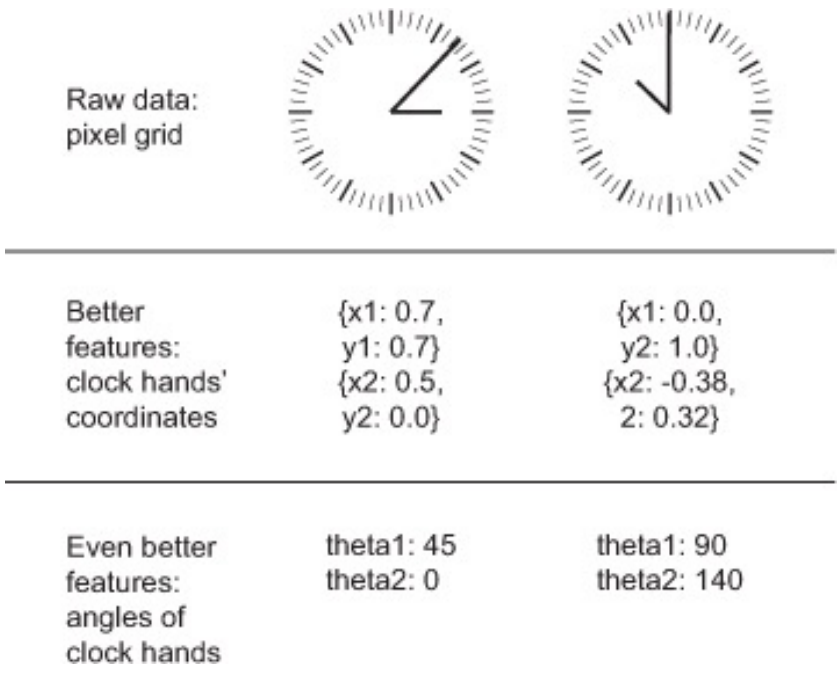


Figure 4.3. Feature engineering for reading the time on a clock

More of an art than a science

Data Preprocessing in Scikit-Learn

- The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.
  - https://scikit-learn.org/stable/modules/preprocessing.html

- Do check it if you want to use Scikit-Learn packages for data preprocessing in the future

Additional References

- Chapter 5 of [T1]

- All code examples in this lecture:
  - Model validation

    https://colab.research.google.com/drive/1LmYPbfxjcWiYXUeX7mO77c94yaL78hP9
  - Data Preprocessing

    https://colab.research.google.com/drive/1UYpZWCl2xB4SL7qE--c7W4OmyPh_0vg4